**08 - Atributos e métodos do NumPy**

Nosso assunto agora são os atributos e métodos com arrays Numpy. Já conhecemos alguns métodos de listas que facilitam bastante o trabalho com elas, e isso também se aplica à essa situação, já que os métodos que conheceremos facilitarão bastante o trabalho com os arrays. Discutiremos um pouco os atributos e métodos mais comuns, mas saiba que existem vários outros que podem ser encontrados na documentação do Numpy.

Nos exemplos usaremos o array dados que criamos anteriormente, um array Numpy bidimensional que possui na primeira linha as informações de quilometragem e na segunda linha os anos de fabricação dos veículos.

dados = np.array([[44410., 5712., 37123., 0., 25757.], [ 2003., 1991., 1990., 2019., 2006.]])

dadosCOPIAR CÓDIGO

array([[44410., 5712., 37123., 0., 25757.], [ 2003., 1991., 1990., 2019., 2006.]])

Os atributos podem ser acessados usando a sintaxe nomedoarray.nomedoatributo. O primeiro que conheceremos é o **shape**. Esse atributo retorna uma tupla com as dimensões do array. Nesse caso, temos a quantidade de linhas na primeira posição e a de colunas na segunda.

dados.shapeCOPIAR CÓDIGO

(2, 5)

Outro atributo é o **ndim**, que retorna o número de dimensões do array. No nosso caso, como temos linhas e colunas, temos duas dimensões.

dados.ndimCOPIAR CÓDIGO

2

O atributo **size** nos mostra o número de elementos de um array. Se o array dados possui duas linhas e cinco colunas, o número de elementos é a multiplicação desses dois valores.

dados.sizeCOPIAR CÓDIGO

10

Para sabermos o tipo de dado dos elementos armazenados no array, algo que já fizemos anteriormente com o Numpy, usamos o atributo **dtype**.

dados.dtypeCOPIAR CÓDIGO

dtype('float64')

Lembrando que se estivéssemos trabalhando com listas, poderíamos ter tipos variados dentro delas, algo que não é possível com arrays Numpy.

O atributo **T** nos retorna o array transposto, ou seja, ele transpõe uma matriz, convertendo linhas em colunas e vice-versa.

dados.TCOPIAR CÓDIGO

array([[44410., 2003.], [ 5712., 1991.], [37123., 1990.], [ 0., 2019.], [25757., 2006.]])

Note que antes tínhamos a quilometragem e os anos como linhas, e agora ambos são colunas. Esse atributo é equivalente à chamada do método transpose():

dados.transpose()COPIAR CÓDIGO

array([[44410., 2003.], [ 5712., 1991.], [37123., 1990.], [ 0., 2019.], [25757., 2006.]])

Dentre os diversos métodos disponíveis, começaremos conhecendo o **tolist()**, que transforma um array Numpy em uma lista do Python. Isso pode ser útil quando precisamos executar algum método ou procedimento que não é possível com os arrays Numpy.

dados.tolist()COPIAR CÓDIGO

[[44410.0, 5712.0, 37123.0, 0.0, 25757.0], [2003.0, 1991.0, 1990.0, 2019.0, 2006.0]]

O método **reshape()** retorna um array contendo os mesmos dados, mas com uma nova forma. Para o primeiro exemplo usaremos uma variável contador que receberá a chamada de np.arange(10), gerando um array Numpy de tamanho 10.

contador = np.arange(10)

contadorCOPIAR CÓDIGO

array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

Agora queremos transformá-lo em um novo array Numpy, mas com 5 linhas e 2 colunas. Note que estamos fazendo uma configuração na qual cabem todos os elementos. Para isso, chamaremos contador.reshape() passando como parâmetro o formato desejado - nesse caso, a tupla (5, 2), representando 5 linhas e 2 colunas.

contador.reshape((5,2))COPIAR CÓDIGO

array([[0, 1], [2, 3], [4, 5], [6, 7], [8, 9]])

O método reshape(), por padrão, usa a indexação da linguagem C. As configurações de indexação são feitas por meio do parâmetro order:

contador.reshape((5, 2), order='C')COPIAR CÓDIGO

array([[0, 1], [2, 3], [4, 5], [6, 7], [8, 9]])

Existe outra forma de indexação, que é a da linguagem Fortran, e podemos configurá-la usando a instrução order='F'.

contador.reshape((5, 2), order='F')COPIAR CÓDIGO

array([[0, 5], [1, 6], [2, 7], [3, 8], [4, 9]])

Isso pode ser útil dependendo do tipo de dado que você está trabalhando, e veremos um exemplo mais prático com os nossos dados futuramente. Note que na indexação da linguagem C os elementos vão sendo colocados nas linhas na mesma ordem em que estavam no array original; Já na indexação da linguagem Fortran, os elementos foram todos distribuídos na primeira posição de cada linha até as 5 serem completadas, e e o restante na segunda posição.

Para exemplificarmos, usaremos duas linhas:

km = [44410, 5712, 37123, 0, 25757]

anos = [2003, 1991, 1990, 2019, 2006]COPIAR CÓDIGO

Criaremos então uma lista info\_carros que será simplesmente a concatenação das duas anteriores.

info\_carros = km + anos

info\_carrosCOPIAR CÓDIGO

[44410, 5712, 37123, 0, 25757, 2003, 1991, 1990, 2019, 2006]

Imagine que já encontramos esses dados "bagunçados", de maneira que fica difícil trabalharmos com eles. O reshape() nos ajuda a solucionar esse tipo de problema. Como os dados são do mesmo tipo, podemos transformá-los em um array Numpy

np.array(info\_carros)COPIAR CÓDIGO

array([44410, 5712, 37123, 0, 25757, 2003, 1991, 1990, 2019, 2006])

Nós automaticamente podemos fazer o reshape() desses dados, por exemplo informando que desejamos um array com 2 linhas e 5 colunas.

np.array(info\_carros).reshape((2, 5))COPIAR CÓDIGO

array([[44410, 5712, 37123, 0, 25757], [ 2003, 1991, 1990, 2019, 2006]])

Repare que nosso resultado é justamente o array dados que tínhamos anteriormente, dessa vez feito a partir de uma lista "bagunçada". Vamos supor que seria mais prático se as informações de cada carro estivessem em linhas individuais - ou seja, cada linha representando a quilometragem e o ano de determinado carro. Para isso, usaríamos a tupla (5, 2), representando 5 linhas e 2 colunas. Mas repare o que acontece quando usamos a indexação padrão da linguagem C:

np.array(info\_carros).reshape((5, 2))COPIAR CÓDIGO

array([[44410, 5712], [37123, 0], [25757, 2003], [ 1991, 1990], [ 2019, 2006]])

Como os dados foram sendo colocados nas linhas de acordo com a ordem original do array, as informações acabaram desconexas. Entretanto, usando order='F' para instruirmos a indexação da linguagem F, os dados serão distribuídos da maneira correta:

np.array(info\_carros).reshape((5, 2), order='F')COPIAR CÓDIGO

array([[44410, 2003], [ 5712, 1991], [37123, 1990], [ 0, 2019], [25757, 2006]])

Agora a primeira linha contém as informações do primeiro carro, a segunda contém as do segundo carro e assim por diante. Esses são alguns "macetes" que vamos aprendendo conforme trabalhamos com dados, e são muito úteis para manipularmos as informações que utilizamos.

O último método que conheceremos nesse vídeo é o **resize()**, que altera a forma e o tamanho do array. Preste atenção nesse ponto, pois incorreremos em um erro propositalmente para ilustrarmos uma situação bastante comum no dia-a-dia de quem trabalha como cientista de dados.

Criaremos uma variável dados\_new que receberá uma cópia do array dados, ou seja, dados.copy().

dados\_new = dados.copy()

dados\_newCOPIAR CÓDIGO

Agora imagine que queremos calcular a quilometragem média desses carros e adicioná-la a uma nova linha do array dados\_new. Para isso, primeiramente usaríamos o resize() passando como parâmetro a tupla (3, 5), representando 3 linhas e 5 colunas.

dados\_new.resize((3, 5))COPIAR CÓDIGO

Como retorno teremos um erro informando que como esses dados são referenciados, é necessário passarmos um parâmetro refcheck-=false, que justamente informa que o Numpy não deverá checar essa referência.

dados\_new.resize((3, 5), refcheck=False)

dados\_newCOPIAR CÓDIGO

array([[44410., 5712., 37123., 0., 25757.], [ 2003., 1991., 1990., 2019., 2006.], [ 0., 0., 0., 0., 0.]])

Com isso, conseguiremos criar a linha extra no array, e poderemos preenchê-las com as novas informações. Para isso, atribuiremos à linha dados\_new[2] o cálculo de quilometragem média com o qual já estamos acostumados.

dados\_new[2] = dados\_new[0] / (2019 - dados\_new[1])

dados\_newCOPIAR CÓDIGO

array([[44410. , 5712. , 37123. , 0. ,

25757. ],

[ 2003. , 1991. , 1990. , 2019. ,

2006. ],

[ 2775.625 , 204. , 1280.10344828, nan,

1981.30769231]])COPIAR CÓDIGO

No próximo vídeo veremos algumas estatísticas descritivas que podemos fazer com arrays Numpy.